Topic 11: Modelling for MIP
(Version of 1st November 2020)

Justin Pearson, Jean-Noël Monette, and Pierre Flener

Optimisation Group
Department of Information Technology
Uppsala University
Sweden

Course 1DL441:
Combinatorial Optimisation and Constraint Programming,
whose part 1 is Course 1DL451:
Modelling for Combinatorial Optimisation

UPPSALA UNIVERSITET

Outline

UPPSALA UNIVERSITET

Outline

UPPSALA UNIVERSITET

Mixed Integer Programming (MIP)

Revisit the slides on mixed integer programming (MIP) of Topic 7: Solving Technologies.

MIP using MiniZinc:

- Many constraint predicates have been equipped with good linear decompositions for use by MIP backends. The performance depends on how well a backend can infer bounds on the variables: see slide 6.
- A MIP backend comes with the MiniZinc distribution, for use with either the open-source MIP solver Cbc or world-class commercial MIP solvers, such as CPLEX Optimizer, FICO Xpress Solver, and Gurobi Optimizer, all free for academic use.

UPPSALA UNIVERSITET

Outline

Big-*M* Formulations

There are modelling idioms in MIP that are referred to as big-*M* formulations; see [Williams, 2013], say.

- The idea is to define a constraint-specific constant *M* that is big enough, but for performance reasons not too big, and to use such constants in order to implement various logical connectives: see next slide.
- The big-*M* constants introduced by a MiniZinc MIP backend depend on how well it can infer upper bounds on the decision variables.
  If one does not specify good bounds on the variables, then solving may be very slow.

Good MIP modelling is very hard: often a PhD per problem.

Example (MIP Idiom for Disequality and Disjunction)

How to model $x \neq y$? Rewrite into $(x + 1 \leq y) \vee (y + 1 \leq x)$. Choose two large constants M_1 and M_2, and introduce a 0-1 variable δ so that $x \neq y$ is modelled as follows:

$$x + 1 \leq y + M_1 \cdot \delta$$
$$y + 1 \leq x + M_2 \cdot (1 - \delta)$$
$$\delta \in \{0, 1\}$$

- If $\delta = 0$, then $x + 1 \leq y$ and $y + 1 \leq x + M_2$, which is not constraining if M_2 is large enough.
- If $\delta = 1$, then $y + 1 \leq x$ and $x + 1 \leq y + M_1$, which is not constraining if M_1 is large enough.

M_1 and M_2 should be large enough to ensure correctness, but as small as possible for performance reasons.
This idiom is not to be used for `all_different`.

UPPSALA UNIVERSITET

Outline

UPPSALA UNIVERSITET

There already are solver-independent modelling languages for mathematical programming, such as AMPL and GAMS. So what are the advantages of using MiniZinc for MIP?

- A unified modelling language that allows one to try CP, SAT, SMT, LS, ... backends and MIP backends on the same instance data, from the same or different models.
- A modelling language with high-level predicates and functions: one may use advanced MIP encoding idioms without even knowing them.

Design a model with a good linear relaxation (LR):

- Ideally, an optimal solution to the LR should have many variables that take integer values.
- This is often impossible, so the aim is to design a model whose LR gives almost-integer solutions.
  Example: Intuitively, if a warehouse `w` is open at 90%, that is `Open[w]=0.9`, then in an optimal solution to the LR one would expect it to be open in an optimal solution to the original problem. Note that this is not always true.
- Use global constraint predicates & constrained functions, add implied constraints, avoid disjunction.
- Try commenting away some symmetry-breaking constraints, as MIP backends currently use the identity definition of `symmetry_breaking_constraint`.

UPPSALA UNIVERSITET

Outline

UPPSALA UNIVERSITET

Outline

Example (The Warehouse Location Problem, WLP)

A company considers opening warehouses at some candidate locations in order to supply its existing shops:

- Each candidate warehouse has the same maintenance cost.
- Each candidate warehouse has a supply capacity, which is the maximum number of shops it can supply.
- The supply cost to a shop depends on the warehouse.

Determine which warehouses to open, and which of them should supply the various shops, so that:

1. Each shop must be supplied by exactly one actually opened warehouse.
2. Each actually opened warehouse supplies at most a number of shops equal to its capacity.
3. The sum of the actually incurred maintenance costs and supply costs is minimised.

WLP: Sample Instance Data

$$\texttt{Shops} = \{\text{Shop}_1, \text{Shop}_2, \ldots, \text{Shop}_{10}\}$$

$$\texttt{Whs} = \{\text{Berlin, London, Ankara, Paris, Rome}\}$$

$$\texttt{maintCost} = 30$$

$\texttt{Capacity} =$

Berlin	London	Ankara	Paris	Rome
1	4	2	1	3

$\texttt{SupplyCost} =$

	Berlin	London	Ankara	Paris	Rome
Shop_1	20	24	11	25	30
Shop_2	28	27	82	83	74
Shop_3	74	97	71	96	70
Shop_4	2	55	73	69	61
⋮	⋮	⋮	⋮	⋮	⋮
Shop_{10}	47	65	55	71	95

WLP Model 3: Variables (reminder)

No automatic enforcement of total-function constraint (1):

`Supply` =

	Berlin	London	Ankara	Paris	Rome
$Shop_1$	$\in$ 0..1	$\in$ 0..1	$\in$ 0..1	$\in$ 0..1	$\in$ 0..1
⋮	⋮	⋮	⋮	⋮	⋮
$Shop_{10}$	$\in$ 0..1	$\in$ 0..1	$\in$ 0..1	$\in$ 0..1	$\in$ 0..1

`Supply[s,w]=1` iff shop `s` is supplied by warehouse `w`.

Redundant decision variables (as in WLP Model 1):

`Open` =

Berlin	London	Ankara	Paris	Rome
$\in$ 0..1	$\in$ 0..1	$\in$ 0..1	$\in$ 0..1	$\in$ 0..1

`Open[w]=1` if and only if warehouse `w` is opened.

WLP Model 3: Objective & Cons. (reminder)

Objective (3):

```
minimize
  maintCost * sum(Open)
  +
  sum(s in Shops, w in Whs)
     (Supply[s,w] * SupplyCost[s,w])
```

Total-function constraint (1):

```
forall(s in Shops)
  (sum(w in Whs)(Supply[s,w]) = 1)
```

Capacity (2) & one-way channelling constraints, combined:

```
forall(w in Whs)(sum(s in Shops)
  (Supply[s,w]) <= Capacity[w] * Open[w])
```

UPPSALA UNIVERSITET

Outline

The Graph Colouring Problem

Given a graph `(V,E)`, colour each vertex so that adjacent vertices have different colours, using at most `n` colours.

Model 1, MIP-style

Variable `C[v,k]` in `0..1` is `1` iff vertex `v` has colour `k`.

1. One colour per vertex:

```
forall(v in V)(sum(C[v,..]) = 1)
```

2. Different colours on edge ends:

```
forall((u,v) in E, k in 1..n)
    (C[u,k] + C[v,k] <= 1)
```

Common pattern in MIP: One needs `0..1` variables to model domains even though one can use integer variables.

Given a graph `(V,E)`, colour each vertex so that adjacent vertices have different colours, using at most `n` colours.

Model 2, MiniZinc/CP/LCG-style

Variable `C[v]` in `1..n` is `k` iff vertex `v` has colour `k`.

1. One colour per vertex: enforced by choice of variables.
2. Different colours on edge ends:

```
forall((u,v) in E)(C[u] != C[v])
```

Give MIP flattening of this model!

UPPSALA UNIVERSITET

Outline

UPPSALA UNIVERSITET

Bibliography


John N. Hooker.
Integrated Methods for Optimization.
2nd edition, Springer, 2012.


H. Paul Williams.
Model Building in Mathematical Programming.
5th edition, Wiley, 2013.